Generating Finite-State Abstractions of Reactive Systems Using Decision Procedures*

Michael A. Colón and Tomás E. Uribe

Computer Science Department, Stanford University
Stanford, CA 94305
colon|uribe@cs.stanford.edu

Abstract. We present an algorithm that uses decision procedures to generate finite-state abstractions of possibly infinite-state systems. The algorithm compositionally abstracts the transitions of the system, relative to a given, fixed set of assertions. Thus, the number of validity checks is proportional to the size of the system description, rather than the size of the abstract state-space. The generated abstractions are weakly preserving for $\forall$CTL* temporal properties. We describe several applications of the algorithm, implemented using the decision procedures of the Stanford Temporal Prover (STeP).

1 Introduction

An attractive method for proving a temporal property φ for a reactive system $\mathcal{S}$ is to find a simpler *abstract system* $\mathcal{A}$ such that if $\mathcal{A}$ satisfies φ, then $\mathcal{S}$ satisfies φ as well. In particular, if $\mathcal{A}$ is finite-state, the validity of φ for $\mathcal{A}$ can be established automatically using a model checker, which may not have been possible for $\mathcal{S}$ due to an infinite or overly large state-space.

There are two obstacles to this approach. First, the construction of $\mathcal{A}$ is often manual and has to be proved correct at a later stage. This process can be error-prone if the proof is not formal, and tedious otherwise. Second, abstractions may not be *fine* enough: if $\mathcal{A}$ is too abstract, $\mathcal{A}$ may not satisfy φ, even if $\mathcal{S}$ does. We address the first problem by automatically constructing an $\mathcal{A}$ that is guaranteed to be a correct abstraction, based on limited user input. We begin to address the second by constructing abstractions quickly: abstractions that are found to be too coarse can be *refined* with little effort and tested again. Thus, finding the right abstraction is an iterative process where the user tests a number of candidate abstractions, possibly guided by feedback from a model checker. The procedure we present can be the basic building block in this process.

As in the procedure of Graf and Saidi [GS97], we use validity checking to generate a finite-state abstraction based on a set of formulas $B = \{b_1, \dots, b_n\}$.

* This research was supported in part by the National Science Foundation under grant CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, ARO under MURI grant DAAH04-96-1-0341, and by Army contract DABT63-96-C-0096 (DARPA).

However, rather than performing an exhaustive search of the reachable abstract states while constructing $\mathcal{A}$, our algorithm transforms $\mathcal{S}$ to $\mathcal{A}$ directly, leaving the exploration of the abstract state-space to a model checker. Thus, the number of validity checks performed by our algorithm is proportional to the number of formulas in B and the size of the representation of $\mathcal{S}$, rather than the size of the abstract state-space. Furthermore, our procedure is applicable to systems whose abstract state-space is too large to enumerate explicitly, but can still be handled by a symbolic model checker [McM93].

The price paid by our approach, compared to [GS97], is that a coarser abstraction may be obtained. However, we offset this by using a richer abstract state-space: the complete boolean algebra of expressions over $B = \{b_1, \ldots, b_n\}$, rather than only the monomials over this set. Our procedure can be seen as a form of abstract interpretation [CC77], with this algebra as the abstract domain.

1.1 Related Work

Our abstraction procedure is related to the work of [GS97] and [SUM96]. The procedure of [GS97] is the closest to ours, as discussed above. In deductive model checking [SUM96], the abstract system and its state-space are generated interactively, using theorem proving, based on the refinement of an initial, maximally abstract system. The refinement proceeds until the property in question can be proved or disproved. That procedure is thus *top-down*, as opposed to the more *bottom-up*, property-independent approaches that this paper and [GS97] propose.[1] In contrast to both [GS97] and [SUM96], we perform validity checking at "compile time," rather than at model check time.

Abstraction frameworks: Theoretical foundations of property-preserving abstraction are presented in [Dam96,LGS+95,CGL94]. We present the necessary results on abstraction in Section 3. Deductive rules for proving simulation and abstraction are presented in [KMP94]. In contrast, our approach is to transform a concrete system into a property-preserving abstract system automatically, obviating the need to prove property preservation for an abstraction given *a priori*.

Approaches based on abstract interpretation [CC77] are presented in, e.g., [CGL94,Dam96,DGG97]. Much of this work is specialized to the case of finite-state systems. We include some simple fairness considerations, a special case of those in the verification rules of [KMP94], which do not appear in most work on abstract interpretation.

Other work uses abstractions that are more explicitly given by the user. For instance, [DF95] applies abstraction and error trace analysis to infinite state systems. The abstraction is generated automatically, given a *data abstraction* that maps concrete variables and operators to abstract ones. [BBM97] uses abstract interpretation to generate invariants and intermediate assertions for fair transition systems. Like ours, their procedure is compositional and automatic,

[1] These procedures can be given a top-down flavor by including in the set B atoms from the temporal formula being verified.

given a suitable abstract domain. Their emphasis is on finding abstraction domains where the reachable state-space can be approximated to produce useful invariants. We, however, are motivated by the need to prove general temporal properties over the abstract system. Nonetheless, our abstractions can be used to generate invariants as well. These invariants can, in turn, be used to generate more precise abstractions.

Over- and under-approximations: Pardo and Hachtel [PH97] present an automatic BDD-based method for symbolic model checking, where the size of BDD's is reduced using over- and under-approximations of subformulas, depending on their polarity. We use polarity in an analogous way. Another approximated BDD-based symbolic model checking procedure is presented in [KDG95], based on the abstract interpretation framework of [Dam96]. These procedures do not change the state-space of the system, but instead approximate the transition relation to produce smaller BDDs.

Dill and Wong-Toi [DW95,Won95] use abstract-interpretation to verify timed safety automata, over- and under-approximating sets of states and next-state relations. This work approximates set operations during model checking, as well as statically approximating the transitions themselves, using methods specialized to real-time systems. The algorithm we propose for over- and under-approximating transitions could be used in similar settings as well.

2 Preliminaries

2.1 Fair and Clocked Transition Systems

Fair transition systems [MP95] are a convenient formalism for specifying finite- and infinite-state reactive systems, using an assertion language based on first-order logic. A fair transition system (FTS) $\mathcal{S} = \langle \Sigma, \Theta, \mathcal{T} \rangle$ is given by a set of *system states* Σ, an *initial condition* Θ, and a set of *transitions* $\mathcal{T}$. Each state in Σ is a valuation of a finite set of typed *system variables* $\mathcal{V}$. If Σ is finite, $\mathcal{S}$ is said to be *finite-state.*

Definition 1 (Assertion). *A first-order formula whose free variables are a subset of $\mathcal{V}$ is an* assertion, *or state-formula, and represents the set of states that satisfy it. For an assertion φ, we say that $s \in \Sigma$ is a φ-state iff $s \models \varphi$, that is, φ holds given the values of $\mathcal{V}$ at s.*

The initial condition Θ is an assertion that characterizes the set of initial states. With each transition $\tau \in \mathcal{T}$ we associate its *transition relation* $\rho_\tau(\mathcal{V}, \mathcal{V}')$, a first-order formula over the system variables $\mathcal{V}$ and a primed set $\mathcal{V}'$, indicating their values at the next state. A transition is *enabled* if it can be taken at a given state. We define *enabled*$(\tau) \overset{\text{def}}{=} \exists \mathcal{V}'. \rho_\tau(\mathcal{V}, \mathcal{V}')$. We define *post*$(\tau, \varphi)$ as the assertion $\exists \mathcal{V}_0. (\rho_\tau(\mathcal{V}_0, \mathcal{V}) \wedge \varphi(\mathcal{V}_0))$, which characterizes the states reachable from φ-states by taking transition τ. As usual, we define φ' to be the result of replacing each free variable x of φ with x'. For a set of expressions E, let $E' \overset{\text{def}}{=} \{\varphi' \mid \varphi \in E\}$.

A *run* of $\mathcal{S}$ is an infinite sequence of states $s_0, s_1, \ldots$, such that $s_0 \models \Theta$ and for all $i \geq 0$, $\rho_\tau(s_i, s_{i+1})$ for some $\tau \in \mathcal{T}$. In this case, we say that τ is *taken* at s_i. Transitions can be labeled as *just* or *compassionate*. A just (or *weakly fair*) transition cannot be continuously enabled without being taken; a compassionate (or *strongly fair*) transition cannot be enabled infinitely often but taken only finitely many times. A *computation* is a run that satisfies all fairness requirements. To ensure that run prefixes can always be extended to an infinite sequence, we assume an *idling transition*, with transition relation $\mathcal{V} = \mathcal{V}'$.

Clocked transition systems [MP96] are an extension of fair transition systems that is intended to model reactive systems with real-time constraints. A clocked transition system (CTS) is a fair transition system $\mathcal{S} = \langle \Sigma, \Theta, \mathcal{T} \rangle$, whose system variables are partitioned into a set of *discrete variables* D and a set of real-valued *clock variables* C. Instead of an idling transition, $\mathcal{T}$ includes a *tick transition*, which is the only transition that can advance time. The progress of time is restricted by a *time-progress condition* Π, an assertion over D and C. The transition relation for *tick* is:

$$\rho_{tick} : \exists \Delta > 0. \begin{pmatrix} (D' = D) \wedge (C' = C + \Delta) \\ \wedge \\ \forall t \in [0, \Delta]. \Pi(D, C + t) \end{pmatrix},$$

where $C' = C + \Delta$ stands for $c_1' = c_1 + \Delta \wedge \ldots \wedge c_k' = c_k + \Delta$, and $\Pi(D, C+t)$ stands for $\Pi(d_1, \ldots, d_j, c_1 + t, \ldots, c_k + t)$, where $D = \{d_1, \ldots, d_j\}$.[2]

We do not impose fairness conditions on the transitions of a clocked transition system. Instead, upper bounds on the time that can pass before an enabled transition is taken can be specified using the time-progress condition. The computations of a CTS are the runs where time grows beyond any bound.

2.2 Temporal Logic

We use linear-time temporal logic (LTL) to express properties of reactive systems. Temporal formulas are built from assertions, boolean operators $(\wedge, \vee, \neg, \rightarrow)$, and temporal operators $(\Box, \Diamond, \mathcal{U}, \mathcal{W})$, as usual. (See [MP95] for details.) LTL properties are part of the universal fragment of CTL*, that is, a subset of $\forall$CTL* [Eme90]. Our procedure applies to the verification of $\forall$CTL* properties, and hence also to LTL.

2.3 Example

Figure 1 presents a fragment of Fischer's real-time mutual exclusion algorithm, as described in [MP96], using the simple programming language of [MP95]. The algorithm assumes uniform positive bounds L and U on the time each process can wait before executing its next statement: an enabled transition must wait at least L and at most U before being taken. If $2L > U$, the algorithm guarantees that both processes are never in their critical sections simultaneously.

[2] Clocked transition systems also contain a *master clock* T, which can only be changed by the *tick* transition; Θ should imply $T = 0$. We will not need T for our example.

$$\textbf{local } x : \{0,1,2\} \textbf{ where } x = 0$$

$$P_1 :: \begin{bmatrix} \ell_0\text{:} & \textbf{await } x = 0 \\ \ell_1\text{:} & x := 1 \\ \ell_2\text{:} & \textbf{skip} \\ \ell_3\text{:} & \textbf{await } x = 1 \\ \ell_4\text{:} & \textbf{critical} \end{bmatrix} \quad \| \quad P_2 :: \begin{bmatrix} m_0\text{:} & \textbf{await } x = 0 \\ m_1\text{:} & x := 2 \\ m_2\text{:} & \textbf{skip} \\ m_3\text{:} & \textbf{await } x = 2 \\ m_4\text{:} & \textbf{critical} \end{bmatrix}$$

Fig. 1. Fischer's mutual exclusion algorithm.

To model the program as a clocked transition system, we introduce two control variables π_1 and π_2, ranging over $\{\ell_0, \ldots, \ell_4\}$ and $\{m_0, \ldots, m_4\}$ respectively, and two clock variables c_1 and c_2. As Θ, we take the assertion $\pi_1 = \ell_0 \wedge \pi_2 = m_0 \wedge c_1 = 0 \wedge c_2 = 0 \wedge x = 0$. We then introduce a transition for each statement, e.g., statement ℓ_1 yields transition τ_{ℓ_1}, with relation

$$\rho_{\ell_1} : \begin{pmatrix} \pi_1 = \ell_1 \;\wedge\; c_1 \geq L \;\wedge \\ \pi_1' = \ell_2 \;\wedge\; c_1' = 0 \;\wedge\; x' = 1 \end{pmatrix} \wedge\; \pi_2' = \pi_2 \;\wedge\; c_2' = c_2 \;.$$

Finally, we take as the time-progress condition $\Pi : c_1 \leq U \wedge c_2 \leq U$, and add the transition *tick*.

Mutual exclusion is expressed by the LTL formula $\Box\neg(\pi_1 = \ell_4 \wedge \pi_2 = m_4)$.

3 Abstraction

Abstraction reduces the verification of a temporal property φ over a *concrete system* $\mathcal{S}$, to checking a related property over a simpler, *abstract system* $\mathcal{A}$. For simplicity, we write $\mathcal{A} \models \varphi$ to indicate that the corresponding property holds for the abstract system.

In the following, we use the notation of [Dam96] whenever possible. Given a set of temporal properties T and two systems $\mathcal{S}$ and $\mathcal{A}$, we say that $\mathcal{A}$ is a *weakly preserving* abstraction of $\mathcal{S}$ for T iff for any $\varphi \in T$, if $\mathcal{A} \models \varphi$ then $\mathcal{S} \models \varphi$. ($\mathcal{A}$ is said to be a *strongly preserving* abstraction if the converse is also true, but we will only use weakly preserving abstractions in this paper.)

Based on the ideas of *abstract interpretation* [CC77], the abstract system can be constructed from an abstract set of states $\Sigma^{\mathcal{A}}$ and a partial order $\preceq$, where $a_1 \preceq a_2$ if a_1 is a "more precise" abstract state than a_2. Such abstractions are often presented in terms of *Galois connections*, where the two posets connected are $(2^{\Sigma}, \subseteq)$ and $(\Sigma^{\mathcal{A}}, \preceq)$, where Σ is the concrete state-space. (see, e.g., [LGS+95,Dam96]). A *concretization function* $\gamma : \Sigma^{\mathcal{A}} \mapsto 2^{\Sigma}$ maps each abstract state to the set of concrete states it represents, and an *abstraction function* $\alpha : 2^{\Sigma} \mapsto \Sigma^{\mathcal{A}}$ maps each set of concrete states to the most precise abstract state that represents it. The pair (α, γ) is a Galois connection iff for all $x \in 2^{\Sigma}$ and all $y \in \Sigma^{\mathcal{A}}$, $\alpha(x) \preceq y$ if and only if $x \subseteq \gamma(y)$. We extend γ to sets of abstract states $S \in 2^{\Sigma^{\mathcal{A}}}$ with $\gamma(S) \overset{\text{def}}{=} \bigcup_{a \in S} \gamma(a)$.

The following abstract domain is often (implicitly) used in deductive verification:

Definition 2 (Assertion-based abstraction). *As the abstract domain Σ^A, choose the complete boolean algebra $\mathcal{BA}(B)$ (using $\wedge^A, \vee^A, \neg^A$) over a finite set of assertions B, where $s_1^A \preceq s_2^A$ iff s_1^A implies s_2^A. Then let $\gamma(f) = \{s \in \Sigma \mid s \models f\}$ and $\alpha(S) = \bigwedge^A \{s^A \in \mathcal{BA}(B) \mid S \subseteq \gamma(s^A)\}$. We call this the* assertion-based abstract domain *with* basis B.

We now have a *Galois insertion* from $(2^\Sigma, \subseteq)$ to $(\Sigma^A, \preceq)$, since $\alpha(\gamma(f)) = f$ for all $f \in \mathcal{BA}(B)$.

Notation: Note that we use $\wedge^A, \vee^A, \neg^A$ for operations in the abstract domain, while $\wedge, \vee, \neg, \rightarrow$ are the usual connectives in the general assertion language.

We will continue to characterize sets of concrete states using assertions, which need not be points in the abstract state-space. For a formula $s^A \in \mathcal{BA}(B)$, we will write $\gamma(s^A)$ to characterize the set of states it represents, rather than simply s^A, to highlight the fact that s^A is an abstract state, while $\gamma(s^A)$ is an assertion (representing a set of concrete states). More formally, $\gamma(s^A)$ is obtained from s^A by replacing $\wedge^A$, $\vee^A$ and $\neg^A$ by $\wedge$, $\vee$ and $\neg$; the boolean variables in s^A, which are elements of B, appear as corresponding subformulas in $\gamma(s^A)$. In this way, the extension of γ to sets can be characterized as $\gamma(S) = \bigvee_{a \in S} \gamma(a)$. For assertions f_1 and f_2, we sometimes write $f_1 \subseteq f_2$ when $f_1 \rightarrow f_2$ is valid.

The correctness of our abstractions is based on the following lemma.

Lemma 1 (Weak Preservation of ∀CTL*—a sufficient condition). *Let B be a finite set of assertions, $\mathcal{S} = \langle \Sigma, \Theta, \mathcal{T} \rangle$, and $\mathcal{A} = \langle \Sigma^A, \Theta^A, \mathcal{T}^A \rangle$. If*

1. *Initial condition: $\Theta \subseteq \gamma(\Theta^A)$ (that is, Θ is over-approximated by Θ^A),*
2. *For each transition $\tau \in \mathcal{T}$ there is a transition $\tau^A \in \mathcal{T}^A$ such that $\rho_\tau \subseteq \gamma(\rho_{\tau^A})$ (that is, τ is over-approximated by τ^A), and*
3. *Fairness: If $\tau^A \in \mathcal{T}^A$ is just (resp. compassionate), then there is a just (resp. compassionate) $\tau \in \mathcal{T}$ such that: (a) $\gamma(enabled(\tau^A)) \subseteq enabled(\tau)$, and (b) $post(\tau, enabled(\tau)) \subseteq \gamma(post(\tau^A, enabled(\tau^A)))$,*

then the abstract system $\mathcal{A}$ is a weakly preserving abstraction of $\mathcal{S}$ for ∀CTL.*

The third requirement limits the fairness constraints that can be imposed on transitions in $\mathcal{A}$. Note that the more fairness constraints $\mathcal{A}$ has, the more ∀CTL* properties it will satisfy. If *(3)* does not hold, only safety properties will be preserved. Note that *(b)* is guaranteed if $\rho_\tau \subseteq \gamma(\rho_{\tau^A})$, in which case *(a)* implies $\gamma(enabled(\tau^A)) \leftrightarrow enabled(\tau)$.

This lemma still holds if the inclusions are valid only for the reachable states of $\mathcal{S}$, i.e., invariants of $\mathcal{S}$ can be used to establish them. More general conditions for simulation and refinement between fair transition systems are presented in [KMP94] as deductive verification rules.

4 Generating Finite-State Abstractions

In the following, let $B = \{b_1, \ldots, b_n\}$ be a fixed assertion basis. We assume we have at our disposal a procedure *checkValid*, which can sometimes decide the validity of assertions: if *checkValid(p)* returns *true*, then p is valid. That is, this validity checker is assumed to be sound, but is not required to be complete.

The workhorse of our abstraction algorithm is a procedure that approximates assertions over $\mathcal{V}$ and $\mathcal{V}'$ as assertions over B and B'. The procedure descends through the boolean structure of the formula, building an assertion to serve as a *context* and keeping track of the polarity of subexpressions until it reaches the atoms. The procedure then over- or under-approximates each atom using an element of $\mathcal{BA}(B \cup B')$.

4.1 Abstracting Atoms

Atoms are abstracted by testing them, in context, against a set of *points* $P \subseteq \mathcal{BA}(B \cup B')$:

$$\alpha_{atom}(+, C, a) = \bigwedge\nolimits^{\mathcal{A}} \{p \in P \mid \mathit{checkValid}((C \wedge a) \to \gamma(p))\} \quad \text{(over-approximation)}$$

$$\alpha_{atom}(-, C, a) = \bigvee\nolimits^{\mathcal{A}} \{p \in P \mid \mathit{checkValid}((C \wedge \gamma(p)) \to a)\} \quad \text{(under-approximation)}$$

Intuitively, the context C indicates that we are only concerned with results that lie within C. Thus, when over-approximating a in context C, we can consider $a \wedge C$ instead, a smaller set. This yields a smaller result, and hence a more precise over-approximation. Similarly, when under-approximating a in context C, we can under-approximate $a \vee \neg C$ instead. This will give a larger result, and hence a better overall under-approximation.

4.2 Abstracting Assertions

We extend α_{atom} to a function α that abstracts assertions as follows:

$$\begin{aligned}
\alpha(\pi, C, a) &= \alpha_{atom}(\pi, C, a), \text{ if } a \text{ is an atom} \\
\alpha(\pi, C, \neg q) &= \neg^{\mathcal{A}} \alpha(\pi^{-1}, C, q), \text{ where } +^{-1} \overset{\text{def}}{=} - \text{ and } -^{-1} \overset{\text{def}}{=} + \\
\alpha(+, C, q \wedge r) &= \textbf{let } \hat{q} = \alpha(+, C, q) \textbf{ in } \hat{q} \wedge^{\mathcal{A}} \alpha(+, C \wedge \gamma(\hat{q}), r) \\
\alpha(+, C, q \vee r) &= \textbf{let } \hat{q} = \alpha(+, C, q) \textbf{ in } \hat{q} \vee^{\mathcal{A}} \alpha(+, C \wedge \neg\gamma(\hat{q}), r) \\
\alpha(-, C, q \wedge r) &= \textbf{let } \hat{q} = \alpha(-, C, q) \textbf{ in } \hat{q} \wedge^{\mathcal{A}} \alpha(-, C \wedge \neg\gamma(\hat{q}), r) \\
\alpha(-, C, q \vee r) &= \textbf{let } \hat{q} = \alpha(-, C, q) \textbf{ in } \hat{q} \vee^{\mathcal{A}} \alpha(-, C \wedge \gamma(\hat{q}), r)
\end{aligned}$$

An assertion f is thus abstracted using $O(|P| \cdot |f|)$ validity checks. The main claim that justifies the correctness of the algorithm is:

Proposition 1. *For assertions C and f,*
$C \to (\gamma(\alpha(-, C, f)) \to f)$ and $C \to (f \to \gamma(\alpha(+, C, f)))$ are valid.

Notice that this algorithm applies to any abstract domain that is a boolean algebra, provided the operations for $\wedge^{\mathcal{A}}$, $\vee^{\mathcal{A}}$, $\neg^{\mathcal{A}}$ and γ are available. Similarly, it applies to any assertion language for which a validity checker is available.

4.3 Abstracting Systems and Properties

Given a concrete transition system $\mathcal{S} = \langle \Sigma, \Theta, \mathcal{T} \rangle$, its abstraction is $\mathcal{A} = \langle \mathcal{BA}(B), \Theta^{\mathcal{A}}, \mathcal{T}^{\mathcal{A}} \rangle$, where $\Theta^{\mathcal{A}}$ is the result of over-approximating Θ, and $\mathcal{T}^{\mathcal{A}}$ is the result of over-approximating each transition relation in $\mathcal{T}$.

The initial context can contain known invariants of $\mathcal{S}$. When abstracting the atoms of an initial condition or the assertions of a temporal property (see below), we test against the set of unprimed points

$$P_U \stackrel{\text{def}}{=} B \cup \{\neg b_i \mid b_i \in B\} \ .$$

For transition relations, we test against the set of mixed points

$$P_M \stackrel{\text{def}}{=} P_U \cup P_U' \cup \{p_1 \rightarrow p_2 \mid p_1 \in P_U \wedge p_2 \in P_U'\} \ .$$

Thus, the algorithm abstracts a transition relation ρ_τ using $O(n^2|\rho_\tau|)$ validity checks, where $n = |B|$. For an assertion f with no primed variables, $O(n|f|)$ validity checks are needed. Enlarging these point sets can increase the quality of the abstraction, as discussed in Section 4.4; however, these relatively small sets sufficed to verify most of the examples in Section 5.

System $\mathcal{A}$ is an n-bit finite-state system. Since $\Theta \subseteq \gamma(\Theta^{\mathcal{A}})$ and $\rho_\tau \subseteq \gamma(\rho_{\tau^{\mathcal{A}}})$ for all $\tau \in \mathcal{T}$, conditions *(1)* and *(2)* of Lemma 1 are satisfied. We satisfy condition *(3)* by propagating the fairness of τ to $\tau^{\mathcal{A}}$ only if we can establish the validity of $\gamma(\mathit{enabled}(\tau^{\mathcal{A}})) \rightarrow \mathit{enabled}(\tau)$. In this case, the two enabling conditions are equivalent.[3] If the basis includes the atoms in the guard of τ, this is guaranteed to be the case. (If an assertion f contains only atoms in B, then its abstraction is equivalent to f, modulo invariants.) In the worst case no fairness carries over and only safety properties of $\mathcal{A}$ (and hence $\mathcal{S}$) can be proved.

A temporal property φ is abstracted by under-approximating the assertions it contains (over-approximating those with negative polarity). This method guarantees that every model of the abstract property corresponds to a model of the concrete one. Thus, if all computations of the abstract system satisfy $\varphi^{\mathcal{A}}$, all computations of the concrete system will satisfy φ. If the basis includes all of the assertions appearing in the property, the property approximation is exact.

4.4 Optimizations

Preserving concrete variables: We allow finite-domain variables of $\mathcal{S}$ to be propagated through to $\mathcal{A}$, leaving it to the model checker to represent them explicitly or encode them as bits. We implement this by having α be the identity on finite-domain subexpressions whose free variables do not appear in the basis. (Note, however, that the algorithm can always be used to abstract finite-state systems to smaller abstract ones.)

[3] In general, the known invariants of $\mathcal{S}$ can be used to establish the conditions of Lemma 1, so the two could differ on unreachable states.

Reducing the test point set: Our implementation includes a few simple strategies for eliminating trivial or redundant test points. For example, if an atom implies b_i, it is unnecessary to test the point $\neg b_i \to b'_j$. Also, if τ does not modify the free variables of b_i, we eliminate the points $\{p \to b'_i \mid p \in P_U\}$.

Enlarging the test point set: There are occasions when additional points must be tested to obtain a sufficiently precise abstraction. For example, ρ_τ may imply $(b_i \wedge b_j) \to b'_k$, but imply neither $b_i \to b'_k$ nor $b_j \to b'_k$. However, we are unwilling to incur the potentially exponential cost of a naive enumeration of such points. Instead, we allow the user to specify additional points to test when specifying the basis. Alternatively, the user may enlarge the basis, but this will in general also increase the time and space used at model-check time.

Conjunctions of literals: When a subexpression consists solely of conjunctions of literals, we eliminate redundant validity checks by testing each point once for the entire subexpression. That is, we terminate the recursion early, since testing the points for each atom will not improve the quality of the abstraction.

4.5 Example

We used the following basis to abstract Fischer's algorithm:

$$\begin{array}{ll} b_1\colon c_1 \geq L & b_4\colon c_2 \geq c_1 \\ b_2\colon c_2 \geq L & b_5\colon c_1 \geq c_2 + L \\ b_3\colon c_1 \geq c_2 & b_6\colon c_2 \geq c_1 + L \end{array}$$

The starting context consisted of assumptions $L > 0, U > 0, U \geq L$ and $2L > U$, and invariants $c_1 \geq 0$ and $c_2 \geq 0$. The initial condition was abstracted to

$$\pi_1 = \ell_0 \wedge \pi_2 = m_0 \wedge x = 0 \wedge \neg b_1 \wedge \neg b_2 \wedge b_3 \wedge b_4 \wedge \neg b_5 \wedge \neg b_6$$

(where we now write $\wedge, \vee, \neg$ rather than $\wedge^A, \vee^A, \neg^A$). Transition ℓ_1 was abstracted to

$$\rho^A_{\ell_1} : \left(\begin{array}{c} \pi_1 = \ell_1 \wedge \pi'_1 = \ell_2 \wedge x' = 1 \wedge \pi'_2 = \pi_2 \wedge \\ b_1 \wedge \neg b'_1 \wedge b'_4 \wedge \neg b'_5 \wedge (b_2 \to \neg b'_3) \wedge (b_2 \to b'_6) \wedge \\ (\neg b_3 \to \neg b'_3) \wedge (\neg b_3 \to b'_6) \wedge (b_4 \to \neg b'_3) \wedge (b_4 \to b'_6) \wedge \\ (\neg b_5 \to \neg b'_3) \wedge (b_6 \to \neg b'_3) \wedge (b_6 \to b'_6) \end{array} \right) .$$

The other transitions were similarly abstracted. (The *tick* transition, which contains quantifiers, was treated as a single literal when abstracted.) With our implementation (see Section 5), the abstract system was generated in 28 seconds, and mutual exclusion was automatically model checked in one second.

5 Experimental Results

We implemented our abstraction procedure using the deductive and algorithmic support found in the Stanford Temporal Prover, STeP [BBC+96]. STeP includes

System	# transitions	Basis size	Abstraction time	Model check time
Bakery	14	3	3s	<1s
Fischer	11	6	28s	1s
Alternating-bit	7	4	14s	<1s
Bounded Retransmission	13	7	70s	4s

Table 1. Abstraction and model check times.

decision procedures for datatypes, partial orders, linear arithmetic, congruence closure and bit-vectors. They are integrated into a general validity checker that is complete for ground formulas, relative to the power of the decision procedures, and can be applied to first-order formulas as well [BSU97]. STeP also includes explicit-state and symbolic LTL model checking for fair transition systems.

We have tested our implementation on a few examples, including two mutual exclusion algorithms and two data-communication protocols. All of these examples are infinite-state: they contain variables whose range is unbounded (integers, lists, and real-valued clocks). For each example, Table 1 gives the size of the basis, the time to generate the abstract system, and the time to model check the properties of interest against the generated abstraction, using STeP's explicit-state model checker.[4] (The concrete systems, properties, bases and abstract systems are available on the web at `http://rodin.stanford.edu/abstraction`.)

The Bakery algorithm is a two-process mutual exclusion algorithm (see, e.g., [MP95]). The property we verify is mutual exclusion, which proves to be particularly easy to establish since it is sufficient to take as a basis the set of assertions that guard transitions.

In the alternating-bit protocol, a sender and a receiver communicate over two lossy channels. The property we verify is that the receiver's list is always a prefix of the list that the sender is transmitting. The basis for this example was found by *trace-based refinement*: starting with the guards of the transitions, we added assertions to the basis in response to abstract counter-examples found by the model checker. We also found it necessary to add a test point so that the validity checker could derive the necessary inductive properties of lists.

The bounded retransmission protocol in, e.g., [HS96,GS97,DKRT97] is an extension of the alternating-bit protocol where a limit is placed on the number of transmissions of a particular item. As with the alternating-bit protocol, we verify the prefix property of the receiver's list. In addition, we verify that the sender and receiver report their status consistently: either they both report `OK`, they both report `NOT_OK`, or the sender reports `DONT_KNOW` and the receiver reports `OK` or `NOT_OK`. To generate the basis, we started with the basis used for the alternating-bit protocol and added the guards of the transitions.

[4] While we recognize that the abstraction times are highly dependent on the speed of the validity checker, we present them to give a feel for how quickly the abstractions can be generated in practice.

While verifying the consistency of the status reports, STeP's model checker discovered an abstract counter-example that uncovered an oversight in our original implementation. If the list to be transmitted is empty, the sender finishes immediately, reporting OK. The receiver, not having received a frame whose last bit was set, assumes the sender aborted transmission and reports NOT_OK. To correct this problem, we require that the list be non-empty, since the bounded retransmission protocol is not designed to transmit empty lists.

6 Conclusions

We have presented a procedure for abstracting transition systems in a compositional manner, using a finite-state abstraction domain. Instead of using theorem-proving to explore the abstract state-space, we use it to abstract the transition relations that describe the system. The abstract state-space can then be explored, explicitly or symbolically, by a model checker.

The procedure provides an alternative method for combining deductive and algorithmic verification. The use of deductive tools makes our procedure applicable to infinite-state systems. The efficiency of the abstraction procedure, and the use of finite-state model checking at the abstract level, gives the procedure a level of automation comparable to that of finite-state algorithmic methods. As with deductive methods, the availability of new decision procedures for particular theories increases the power of the algorithm.

The choice of the abstraction basis B can be based on the user's understanding of the system, analogous to the use of intermediate assertions in deductive verification. The procedure is completely automatic once this basis is chosen, and its efficiency allows for various alternatives to be quickly tested. However, techniques for the generation (manual and automatic) of the abstraction basis remain to be tested and explored.

Acknowledgements: We thank Nikolaj Bjørner, Zohar Manna, Hassen Saidi and Henny Sipma for their feedback and comments.

References

[AH96] R. Alur and T.A. Henzinger, editors. *Proc. 8th Intl. Conference on Computer Aided Verification*, vol. 1102 of *LNCS*. Springer-Verlag, July 1996.

[BBC+96] N.S. Bjørner, A. Browne, E.S. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In Alur and Henzinger [AH96], pages 415–418.

[BBM97] N.S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997. Preliminary version appeared in 1st *Intl. Conf. on Principles and Practice of Constraint Programming*, vol. 976 of LNCS, pp. 589–623, Springer-Verlag, 1995.

[BSU97] N.S. Bjørner, M.E. Stickel, and T.E. Uribe. A practical integration of first-order reasoning and decision procedures. In 14th *Intl. Conf. on Automated Deduction*, vol. 1249 of *LNCS*, pages 101–115. Springer-Verlag, July 1997.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4th *ACM Symp. Princ. of Prog. Lang.*, pages 238–252. ACM Press, 1977.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Trans. on Prog. Lang. and Systems*, 16(5):1512–1542, September 1994.

[Dam96] D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, July 1996.

[DF95] J. Dingel and T. Filkorn. Model checking of infinite-state systems using data abstraction, assumption-commitment style reasoning and theorem proving. In Wolper [Wol95], pages 54–69.

[DGG97] D.R. Dams, R. Gerth, and O. Grümberg. Abstract interpretation of reactive systems. *ACM Transactions on Prog. Lang. and Systems*, 19(2):253–291, 1997.

[DKRT97] P.R. D'Argenio, J.P. Katoen, T. Ruys, and G.T. Tretmans. The bounded retransmission protocol must be on time! In *3rd TACAS Workshop*, vol. 1217 of *LNCS*, pages 416–432. Springer-Verlag, 1997.

[DW95] D.L. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In Wolper [Wol95], pages 409–422.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, pages 995–1072. Elsevier Science Publishers (North-Holland), 1990.

[Gru97] O. Grumberg, editor. *Proc.* 9th *Intl. Conference on Computer Aided Verification*, vol. 1254 of *LNCS*. Springer-Verlag, June 1997.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In Grumberg [Gru97], pages 72–83.

[HS96] K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *Formal Methods Europe*, pages 662–681, March 1996.

[KDG95] P. Kelb, D. Dams, and R. Gerth. Practical symbolic model checking of the full μ-calculus using compositional abstractions. Technical Report 95/31, Eindhoven University of Technology, The Netherlands, October 1995.

[KMP94] Y. Kesten, Z. Manna, and A. Pnueli. Temporal verification of simulation and refinement. In *A Decade of Concurrency*, vol. 803 of *LNCS*, pages 273–346. Springer-Verlag, 1994.

[LGS$^+$95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Pub., 1993.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[MP96] Z. Manna and A. Pnueli. Clocked transition systems. Tech. Report STAN-CS-TR-96-1566, Computer Science Department, Stanford University, April 1996.

[PH97] A. Pardo and G. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In Grumberg [Gru97], pages 12–23.

[SUM96] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In Alur and Henzinger [AH96], pages 208–219.

[Wol95] P. Wolper, editor. *Proc.* 7th *Intl. Conference on Computer Aided Verification*, vol. 939 of *LNCS*. Springer-Verlag, July 1995.

[Won95] H. Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems*. PhD thesis, Computer Science Department, Stanford University, March 1995. Tech. Report CS-TR-95-1546.